Evaluating Embedded Parsers

Will Fitzgerald and Christopher K. Riesbeck
The Institute for the Learning Sciences, Northwestern University
Evanston, Illinois, USA 60201
{fitzgerald, riesbeck}@ils.nwu.edu

Abstract

We need to build parsers that are robust in the face of many types of parsing challenges, and evaluate them for the accuracy and transparency of their results, and how well the parser scales up to practically-sized problems. We describe a number of ways to measure accuracy, transparency and scale-up, in the context of evaluating the parser in the Casper system, a tutorial for training customer service representatives in complex problem solving.

Introduction

Natural language processing (NLP) began with the building of software systems that could interpret human language and turn it into something useful—for example, the machine translation proposals of the 40's and 50's (e.g., Weaver 1955) were put forth with the very practical goal of building systems that could, for example, translate Russian chemistry texts into English. As research into NLP progressed, however, the difficulty of doing NLP became very apparent, and, in many cases, NLP research was abandoned.

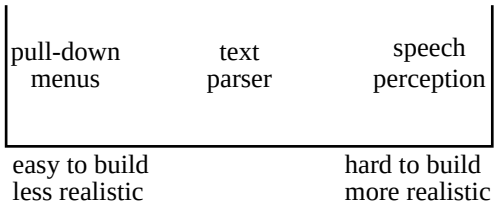
However, it is still often the case that an effective NLP system would be a very useful system to have, and many approaches have been proposed to provide NLP, especially in limited domains. In this paper, we discuss evaluation metrics for parsers that have been created to be built into, or *embedded* into application programs). Specifically, we'll look at parsers that were created for the purpose of carrying on conversations with simulated agents.

Communication with simulated agents

A common task for an embedded parser is to allow a user of the application program to communicate with a simulated agent. Consider the Casper system, for example. Casper was built to train customer service representatives of a British water utility, North West Water, plc (Kass 1994). The major task of a customer service representative is to converse with customers on the telephone about customer inquiries and problems—for example, problems with water quality. Casper is a simulation program—novice customer service representatives have the opportunity to talk to simulated customers about problems the customers face.

An important question is how the communication between the student and the simulated customer will take place. Figure 1 shows a spectrum of possibilities: from pull-down menus to text parsers to speech perception.

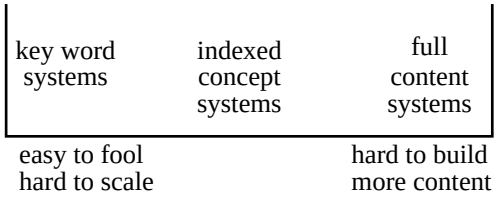
Figure 1: The spectrum of interface possibilities.



An interface using speech perception would be very difficult to build but add to the realism of a simulation; communicating with pull-down menus may be easy to build, but impinge on the naturalness of the simulation. A possible compromise is to use a text parser: the user types in what he or she wants to say, and embedded parser determines what the student meant.

Text parsers, too, vary on a spectrum of possibilities, based on the amount of knowledge or content which support them. Key word systems have little or no content support them: statistical regularities are typically used for parsing (Salton & McGill 1983). Full content systems, such as Direct Memory Access Parsing (Martin 1989, 1990; Fitzgerald 1994a) require relatively complete representations for parsing.

Figure 2: The spectrum of text parser possibilities.



For the Casper project, we elected to build a parser based on indexed concepts (Fitzgerald 1994b). In a sys-

tem using indexed concept parsing, each potential parse is tagged with sets of index concepts. Index concepts can be arranged in a taxonomic hierarchy; each index concept is tagged with phrasal patterns. For example, the index concept `Water-Bits` might be tagged with the phrasal patterns "bits" or "particles." Reading "bits" in the input sentence "Can you describe what the bits look like?" will activate the `Water-Bits` index concept. Activating an index concept means that it will provide positive evidence for each potential parse with which it has been tagged, and negative evidence for each potential parse with which it is not been tagged. Hierarchical relationships between index concepts are honored, as well: If `Water-Bits` are an instance of `Water-Quality-Problem`, then activating also `Water-Bits` activates `Water-Quality-Problem`.

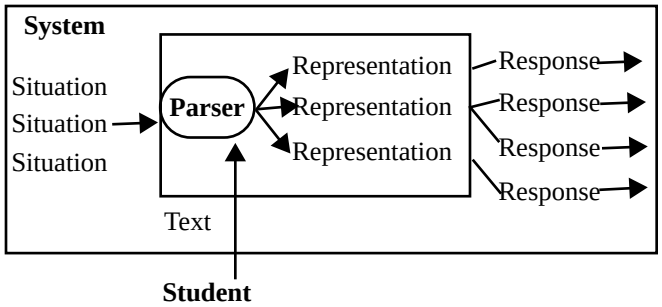
The advantage of indexed concept parsing is that going about the task of creating content to support an embedded parser is simplified—the index concepts have no structure (in the limit), nor do the phrasal patterns associated with them (in the limit).

In building a new technology, we want to answer the question, Does the technology work? Does it do what it was built to do? Before we can measure this, however, we need to understand exactly what a parser is built to do.

A model of embedded parsing

In Casper, a novice customer service representative will communicate with simulated customers. As the student communicates, a tutorial system intervenes. As with any such program, the number of interventions is limited—the tutor will be built to respond in certain situations, but not in others. So, a key feature of such systems is that there is a limited (even if large) number of responses the system can process. In general, we can build a model of an embedded parser (Figure 3 shows a diagram).

Figure 3: Model of an embedded parser



An embedded parser is built to transform *text* that a *user* enters in a particular *situation*, for which the system has a set of *possible response*s. The system is built to turn that text into a *representation* or set of representations that the system can use to make the appropriate responses.

In the case of the Casper tutor, a user might be in the situation of troubleshooting a (simulated) customer's water quality problem, specifically (for example) trying to determine exactly what the problem is that the customer is experiencing. Among the tutor's possible responses is to intervene if the student is asking leading questions (for example, "Do you have black bits in the water?") and tell the student to ask non-leading questions ("Can you describe the water to me?"). Among the representations that the parser might produce are the representations for the questions "Do you have black bits in the water?" and "Can you describe the water to me?"

So, given this model of embedded parsing, to evaluate a parser embedded in an application program, we want to ask:

- Does the parser produce the right representations? That is, is the parser accurate?
- Does the parser promote the sense that a conversation is taking place? That is, is the parser transparent?
- Is the parser capable of adequately handling the varieties of texts, users, situations and possible responses it might meet? That is, is the parser scalable?

Measuring Accuracy

To measure the accuracy of a parser presumes we know the answer to the question, what has the parser been built for? For example, parsers built as general, off the shelf tools such as the CLARE system (Alshawi et al 1992) or the Alvey NL Tools (Briscoe, et al. 1987), might define accuracy as coverage: that is, how much of naturally occurring language the parser is likely to be able to process. A typical statement of coverage, perhaps, comes from a description of the natural language processing system in the Cyc project: "The Syntax module can properly handle about 75% of the sentences found in the news stories of a typical issue of the newspaper *USA Today* ... by 'properly' here we mean that the output is as good as what our knowledge entered independently come up with." (Guha and Lenat 1994, p. 138), and from a description of the Alvey NL Tools: its grammar is "in principle able to analyse 96.8% of a corpus of 10,000 noun phrases taken from a variety of corpora (Carroll 1994, footnote 2)." Parsers might be built for more immediate tasks, such as text categorization or information extraction. For example, the Message Understanding Conferences (MUC) bring together a variety of research systems to compare their abilities to read articles (for example, about terrorist episodes) and fill in a template about the article, for example, who were

the victims, the perpretrators, etc. (Jacobs and Rau 1993; Soderland and Lehnert 1994).

But our task is not syntactic coverage, information extraction or text categorization. Rather, we are describing parsers embedded into application programs. Fig. 1 describes a model of such a parser. There are three inputs to a parser: a text, a user and a situation. The output of the parser should be a set of representations that accurately model the intentions of the user in the situation, as expressed by the text.

To measure the accuracy of a parser, we assume the existence of an *oracle* that can return just those representations that do accurately model the intentions of the user in a situation as expressed by the text—assuming the representations exist that do so model the user's intentions. If we have such an oracle, we can compare the output of the parser with the output of the oracle. We will call the representations that an oracle would produce *oracular representations*, and the representations produced by the parser, *parsed-to representations*. Let us call a representation that is both a parsed-to representation and an oracular representation a *relevant representation*.

Recall and Precision

There are a variety of ways of comparing the output of the oracle and the output of the parser. Two of the most commonly used are *recall* and *precision* (Salton and McGill 1983; Jacobs and Rau 1993; Soderland and Lehnert 1994). Recall is the ratio of relevant representations to oracular representations. We want parsers that have high recall—this means that the parser produces the representations that the idealized oracle produces (although it may also produce other representations that the oracle wouldn't). In equation form:

$$\frac{|P \cap O|}{|O|} \qquad \text{(EQ 1)}$$

where P is the set of parsed-to representations and O is the set of oracular representations.

Ideally, we want the parser to produce just those representations that the oracle would produce. Precision is a measure of this: it is the ratio of relevant representations to all of the representations produced by the parser. In equation form:

$$\frac{|P \cap O|}{|P|} \qquad \text{(EQ 2)}$$

again, where P is the set of parsed-to representations and O is the set of oracular representations.

Where does the oracle come from?

An important question to ask is where the oracle comes from. Because the oracle is an idealized construct, we can only approximate one. One approach is to avoid approximating the oracle altogether, and to use information theoretic measures of the effect of different information sources on the probability that a given hypothesis, $r \in O$, is true (for example, Rosenfeld 1994). Another approach, exemplified by the MUC evaluations, is to approximate the oracle by enumerating the correct answers for a given sample of inputs, and assume that these results project to the general population of inputs (Jacobs and Rau 1993). This is the approach we will take in this work.

***Perfect* and *acceptable* recall**

In the model of embedded parsing given above, the output of the parser is a representation or set of representations. It may be that, for the purposes of the application program, it would be acceptable for an oracular representation to appear within a set of representations, of a certain size. For example, in the Casper system, the student can pick from the top seven best results of the parser. As long as the oracular representation is within the top n results, we can call it a relevant representation. Figure 4 shows the results from the Casper parser from the text, "Describe the particles, please." The oracular match, "What kind of bits are in you water?" is shown in bold face. Because this sentence appears in the top 7 results, we call this an *acceptable match*; because it is also the best match returned by the parser, we say it is a *perfect match*.

Figure 4: Results from parsing "Describe the particles, please"

Parse: Describe the particles, please

Results:

What kind of bits are in your water?
Can you describe the problem?
Can you draw some water and tell me how it smells?
Can you draw some water and tell me how it tastes?
Can you draw some water and tell me how it looks?
Can you run the water and tell me what you see?
Can you run the cold tap and tell me what you see?

With these definitions in mind, we can also define perfect recall and acceptable recall. Perfect recall is the recall ratio when we describe a representation as relevant if it is the best (or only) match produced by the parser. Acceptable recall is the recall ratio when the

representation is relevant as long as it appears in the top *n* matches produced by the parser.

Measuring transparency

Recall and precision are measures of the accuracy of a parser. We would also like to measure how *transparent* a parser is. Following Heidegger, Winograd and Flores (1987) discuss the *readiness to hand* of tools. A tool is ready to hand if it becomes transparent in its use. That is, we don't notice a hammer when we're pounding a nail because we focus on the goal of getting the nail in the wood. If the hammering fails, we might then examine the hammer—perhaps the flat end has become worn, or there is some other problem. The hammer becomes unready to hand; while examining it, we cannot be pounding. Similarly, a parser is ready to hand, or transparent, when the user of the parser does not have to wonder about the tool and its workings, but can concentrate on achieving the communicate goals he or she has. Two characteristics of a parser that will affect transparency are *time to parse* and *number of negotiations required*. These characteristics are defined below.

Measuring speed

Accuracy measurements affect transparency, but there are many different reasons that a parser could become non-transparent. A parser will tend to become less transparent as the time it takes to return its results increases. The amount of time it takes for a parser to return its results for a specific interaction we will call the *time to parse*.

In the natural language parsing literature, time to parse are typically measured in one of two ways. In the first way, the algorithmic computational complexity of the parsing algorithms used are examined. The advantage of this approach is that a theoretical upper bound on the parse time can be given. Unfortunately, most parsing algorithms have upper bounds that are cubic to the length of the input string (that is, an estimate of how long it will take to parse a sentence of *n* words will be dominated by some function of n^3). Empirically, though, parsers tend to do much better than this, and so even researchers who are of a theoretical bent are proposing empirical testing of different parsers (Slocum 1981, Carroll 1994). For parsers in interactive programs, perhaps the best measure of parse time is *perceived average wall time*: that is, asking the users of the system whether, on average, the parser was fast enough in returning its results.

Measuring negotiation

It is often the case that an interactive program will allow the user to commit to or to reject the results of a parse, and, if the user rejects the parse, to try again to produce an acceptable parse. This is analogous to a human conversation, in which one person says something, call it A, and the other person asks whether by A the first person meant B (a paraphrase of A). The first person can agree, or try again. We'll assume that the fewer number of turns the user needs to negotiate the meaning of the user's input, the more transparent the parser will be. Measuring the how long it takes on average to come to an agreement we'll call *negotiation length*.

We have developed two measurements of negotiation, both of which make use of the student's being able to make multiple attempts before committing to a result. One important fact is that a student must (unless he or she gives up on the entire tutorial) commit to some result at some time. The first measurement we call the *first strike percentage*. This measures the percentage of times the parser returns a result that is accepted by the user on the first attempt. We can also measure how many attempts it takes for a student to reach a result, and take the average this over all the attempts. This we'll call the *average path length*; the closer the average path length is to 1.0, the more transparent it is (an interface with a first strike rate of 100% would have an average path length of 1.0).

Measuring scale-up

One of the most important questions to ask of any parsing technology, especially ones built on concepts and techniques in artificial intelligence, is whether it will scale up to large problems (Schank 1991). Previously, we described a spectrum of text parsers, from key word parsers, to indexed concept parsers, to full content parsers. Key word systems tend to fail as they are scaled up to large problems because they rely on the statistical correlation of text to underlying meaning. Factors such as synonymy and ambiguity tend to lower the correlation. Full content parsers, on the other hand, tend to fail as they are scaled up because they are difficult to build; the underlying conceptual representations required can be arbitrarily hard.

Our hope, as we developed an indexed concept parser for the Casper project, was that indexed concept parsing would provide most of the advantages of both key word systems and full content parsers, that is, by creating minimalist representations, we could solve (some of) the problems key word systems have with synonymy and ambiguity without having to create large numbers of

articulated representations required to do full content parsing.

By creating an index concept parser, we wanted to build a parser that would scale up. But the question remains, how would we measure this?

We have hinted that there are two concerns related to scale up. One concern is with the development cost of scaling up: is it possible to create the representations need to support parsing? The other concern is with the complexity of the representations of the application program. Assuming that representations can be built for parsing, how well will the parser behave (for example, on recall measures) as the underlying complexity increases?

Practically, though, we only need measure scale-up for a particular implementation of a parser in an application program. In this case, the complexity of the underlying representations is known. In terms of the model of embedded parsing given before, we know the range of situations and responses which the parser will encounter (although we probably do not know the range of texts and users the parser will encounter).

Then, measuring scale-up becomes a qualitative answer to the following questions:

- Is the cost of creating the parser acceptable?
- Is recall acceptable for the entire range of situations and possible responses?

Our suspicions are that key word systems will have acceptable development costs but low recall for medium to large (open-ended) systems; that full content parsers will have high development costs but low recall for small to medium size systems, and impossibly high development costs for large systems; and that indexed concept parsing will have acceptable costs and recall rates for small to medium systems, but unacceptable recall rates for large systems.

Evaluation of the Casper parser

Having mentioned these measurements in the context of Casper, we now describe the actual results of using some of these measures on the use of the parser in Casper. The following tables show the results of the beta testing of Casper on 12 customer service representative trainees who were engaged in up to six conversations with simulated customers.

Table 1 shows relevance measures. The percentage of acceptable matches was 84.1% in the beta testing. We hope to use the results of the beta testing to improve the parser, particularly in adding synonyms for indexed concepts, representations of different ways to achieve the same communicate, goal and handling anaphora.

TABLE 1. Accuracy measures in Casper (recall)

Accuracy measures (N=391)	i	%
Percentage of acceptable matches (set size = 7; best=100%)	329	84.1%
Percentage of perfect matches	273	69.8%

Table 2 shows transparency measures. The first strike rate for the parser, even in the strict version of this, was higher than the menus, and the average path length was longer—both giving evidence that the type-in box with the parser is preferable to the hierarchical menus.

TABLE 2. Transparency measures in Casper (strike rate)

Strike rate	n	i	%
First strike rate, hierarchical menus (best=100%)	260	179	68.8%
Loose first strike rate, parser	391	322	82.4%
Strict first strike rate, parser	391	270	69.1%

Table 3 shows average path length measures. It is interesting to compare the frequency with which the parser and the hierarchical menus were used on the first attempt: many more interactions were started with the parser than the menus (391 and 260, respectively); this concurs with user reports that they preferred to use the parser. The average path length for the parser is less than the average path length of the hierarchical menus. This indicates that the students were able to say what they wanted to say in fewer tries using the parser.

TABLE 3. Path length measures in Casper (path length)

Average path length	n	length
Average path length, hierarchical menus (best=1.00)	260	1.75
Average path length, parser	391	1.28

Conclusions

Historically, parsers have been created to prove some theoretical point. But building parsers for practical applications will affect how we evaluate them. We need to build parsers that are robust in the face of many types of parsing challenges, and evaluate them for their accuracy and transparency, and how well they scale up for practical systems.

Acknowledgments

This work was supported in part by the Defense Advanced Research Projects Agency, monitored by the Office of Naval Research under contracts N00014-91-J-4092 and N00014-90-J-4117. The Institute for the Learning Sciences was established in 1989 with the support of Andersen Consulting, part of The Arthur Andersen Worldwide Organization. The Institute receives additional support from Ameritech and North West Water Group plc, Institute Partners, and from IBM.

References

Alshawi, H., Carter, D., Crouch, R., Pulman, S., Rayner, M. & Smith, A. (1992). CLARE: A Contextual Reasoning and Cooperative Response Framework for the Core Language Engine. Cambridge, UK: SRI International.

Briscoe, E., Grover, C., Boguraev, B. & Carroll, J. (1987). A formalism and environment for the development of a large grammar of English. In Proceedings of 10th International Joint Conference on Artificial Intelligence, (pp. 703-708).

Carroll, J. (1994). Relating complexity to practical performance in parsing with wide-coverage unification grammars. In the Proceedings of the Association for Computational Linguistics, (to appear).

Fitzgerald, W. (1994a). Direct memory access parsing in the Creanimate biology tutor. In *Proceedings of the Tenth Conference on Artificial Intelligence for Applications*, (pp. 467-468). Los Alamitos, CA: IEEE Computer Society Press.

Fitzgerald, W. (1994b). Indexed concept parsing for interactive tutors. *Proceedings of the Active Natural Language Workshop of the AAAI Spring Symposium* 1994.

Guha, R. V. & Lenat, D. B. (1994). Enabling agents to work together. Communications of the ACM, 37(7), 127-142.

Jacobs, P. S. & Rau, L. F. (1993). Innovations in text interpretation. Artificial Intelligence, 63(1), 143-191.

Kass, A, (1994). The Casper Project: Integrating simulation, case presentation, and Socratic tutoring to teach diagnostic problem solving in complex domains. Technical Report 51. The Institute for the Learning Sciences, Northwestern University, Evanston, IL.

Martin, C. E. (1989). Case-based parsing and Micro-DMAP. In Inside Case-based Reasoning, C.K. Riesbeck and R. C. Schank, eds. (pp. 319–392). Lawrence Erlbaum Associates, Hillsdale, N.J.

Martin, C. E. (1990). *Direct Memory Access Parsing*. Ph.D. dissertation, Yale University, New Haven, CT.

Rosenfeld, R. Adaptive Statistical Language Modeling: A Maximum Entropy Approach. Technical Report CMU-CS-94-138, Carnegie Mellon University, Pittsburgh, PA.

Salton, G. & McGill, M. J. (1983). *Introduction to Modern Information Retrieval*. McGraw-Hill.

Schank, R. (1991). Where's the AI? AI Magazine, 12:4, pp. 38–49.

Slocum, J. (1982). A practical comparison of parsing strategies. In Proceedings of Proceedings of the 19th Annual Meeting of the Association for Computational Linguistics, (pp. 1-6).

Soderland, S. & Lehnert, W. (1994). Corpus-driven knowledge acquisition for discourse analysis. In the Proceedings of the Twelfth National Conference on Artificial Intelligence, (to appear).

Weaver, W. (1955). Translation. In W. N. Locke & A. D. Booth (Eds.), *Machine Translation of Languages* (pp. 15-23). London: John Wiley and Sons.

Winograd, T., & Flores, F. (1987). *Understanding Computers and Cognition: A New Foundation for Design*. Reading, MA: Addison-Wesley.